UNITED STATES PATENT AND TRADEMARK OFFICE

mw

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/642,360	08/15/2003	Adam G. Wolff	LZLO-01008US0	1630

28554 7590 10/31/2006

VIERRA MAGEN MARCUS & DENIRO LLP
575 MARKET STREET SUITE 2500
SAN FRANCISCO, CA 94105

EXAMINER	
WEI, ZHENG	
ART UNIT	PAPER NUMBER
2192	

DATE MAILED: 10/31/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/642,360	WOLFF ET AL.	
	Examiner	Art Unit	
	Zheng Wei	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *15 August 2003*.

2a)□ This action is **FINAL**. 2b)☒ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-37* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-37* is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *15 August 2003* is/are: a)☒ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)□ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____.

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) □ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) □ Notice of Informal Patent Application
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *08/15/2003*.	6) □ Other: _____.

DETAILED ACTION

1. This office action is in response to the application filed on 08/15/2003.

2. Claims 1-37 are pending and have been examined.

Priority

3. The priority date for this application is 08/15/2003. No continuing data and foreign applications are related to this application.

Information Disclosure Statement

4. The information disclosure statement filed 08/15/2003 has been placed in the application file and the information referred to therein has been considered.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 1-5, 7, 13-15, 18, 19, 21, 26, 27, 29 and 35 are rejected under 35 U.S.C. 102(b) as being anticipated by Steele (Guy L. Steele, Common Lisp the language, 2nd edition)

Claim 1:

Steele discloses a method to provide for evaluating expressions, comprising:

- Receiving code for a program, said code includes one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program (see for example, section 5.3.3, Control of Time of Evaluation, [Special Form], p.11, line 11, "situation" and related text); and
- Automatically providing additional functionality to said code for said program, said additional functionality evaluates said one or more expressions during execution of said program at one or more times specified by said one or more markers (see for example, section 5.3.3, Control of Time of Evaluation, p.11, line 14, "interpreter", "compiler"),

Claim 2:

Steele further discloses a method according to claim 1, wherein: said one or more markers specify when said one or more expressions should be evaluated during execution of said program independent from a context of where said expressions are used (see for example, section 5.3.3, Control of Time of

Evaluation, p.11, line 8, "...executed only at compile time, only at load time or when interpreted but not compiled.")

Claim 3:

Steele also discloses a method according to claim 1, wherein: said markers can indicate that a particular expression should be evaluated immediately, once or always (see for example, section 5.3.3, Control of Time of Evaluation, p.11, lines 12-15, "Each *situation* must be a symbol, either *compile, load or eval. eval* specifies that the interpreter should process the body. *compile* specifies that compiler should evaluate the body at compile time in the compilation context").

Claim 4:

Steele further discloses a method according to claim 3, wherein: failure of a marker to indicate that a particular expression should be evaluated immediately or once defaults to an indication indicate that said particular expression should be evaluated always (see for example, section 5.3.3, Control of Time of Evaluation, p.11, lines 32-34, "If the form is not an eval-when form...perform normal compiler processing of the form").

Claim 5:

Steele further discloses a method according to claim 1, wherein:

- Said one or more expressions are constraints for variables (see for example, section 5.3.3, Control of Time of Evaluation, p.12, lines 1-4, example code, "lambda (stream char)", "declare (ignore char)"; and
- Said step of automatically providing additional functionality to said code includes adding codes that creates an object for each constraint, adds functions to said object that sets said variables, and adds functions that sets dependencies for said expressions (see for example, section 5.3.3, Control of Time of Evaluation, p.12, lines 1-5, example code and related text explanation, "This causes the call to set-macro-character to be executed in the compiler's execution environment, thereby modifying its reader syntax table.").

Claim 7:

Steele also discloses a method according to claim 1, wherein: said step of automatically providing additional functionality to said code includes compiling said code (see for example, section 5.3.3, Control of Time of Evaluation, p.11, lines 14-15, "*compile* specifies that compiler should evaluate the body at compile time in the compilation context").

Claim 13:

Steele discloses a method to provide for evaluating expressions, comprising:

- Receiving code for a program, said code includes one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program; (see for example, section 5.3.3, Control of Time of Evaluation, [Special Form], p.11, line 11, "situation" and related text) and
- Evaluating said one or more expressions during execution of said program at times specified by said one or more markers (see for example, section 5.3.3, Control of Time of Evaluation, p.11, line 14, "interpreter", "compiler"),

Claim 14:

Steele further discloses a method according to claim 13, wherein: said one or more markers specify when said one or more expressions should be evaluated during execution of said program independent from a context of where said expressions are used (see for example, section 5.3.3, Control of Time of Evaluation, p.11, line 8, "...executed only at compile time, only at load time or when interpreted but not compiled.").

Claim 15:

Steele further discloses method according to claim 13, wherein: said markers can indicate that a particular expression should be evaluated immediately, once or always (see for example, section 5.3.3, Control of Time of Evaluation, p.11, lines

12-15, "Each *situation* must be a symbol, either *compile, load or eval. eval* specifies that the interpreter should process the body. *compile* specifies that compiler should evaluate the body at compile time in the compilation context").

Claim 18:

Steele discloses a method to provide for evaluating expressions, comprising:

- Accessing code that includes an expression defining a first variable, said expression is dependent on a changeable item (see for example of implementation a *defun* form, p.14, line 14, a first variable "x" and expression); and
- Compiling said code, said step of compiling said code adds additional functionality to said code, said additional functionality evaluates said expression when said item changes and updates said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25 and related text explanation).

Claim 19:

Steele further discloses a method according to claim 18, wherein:

- Said expression is part of a constraint for said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 17, "compiler::notice-function 'bar ' (x)")

- Said step of compiling includes creating an object for said constraint, adding a first function to said object that sets said first variable; determining dependency of said expression and adding a second function for said dependency (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25, "lambda (x)", "lambda () (+ x 3)" and related text explanation).

Claim 21:

Steele further discloses a method according to claim 18, wherein: said code includes a marker for said expression, said marker specifies when said expression should be evaluated during execution of said code (see for example code of implementation to expand a defun form into eval-when, p.14, lines 16-25, marker ":compile-toplevel" and related text explanation).

Claim 26:

Steele discloses a method to provide for evaluating expressions, comprising:

- Receiving code that includes an expression defining a first variable, said expression is dependent on a changeable item (see for example of implementation a *defun* form, p.14, line 14, a first variable "x" and expression); and
- Automatically providing additional functionality to said code, said additional functionality evaluates said expression when said item changes

and updates said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25 and related text explanation).

Claim 27:

Steele further discloses a method according to claim 26, wherein:

- Said expression is part of a constraint for said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 17, "compiler::notice-function 'bar ' (x)"); and
- Said step of automatically providing includes creating an object for said constraint, adding a first function to said object that sets said first variable, determining dependency of said expression and adding a second function for said dependency to said object (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25, "lambda (x)", "lambda () (+ x 3)" and related text explanation).

Claim 29:

Steele discloses a method according to claim 26, wherein: said code includes a marker for said expression, said marker specifies when said expression should be evaluated during execution of said code (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25, marker ":compile-toplevel" and related text explanation).

Claim 35:

Steele further discloses one or more processor readable storage devices according to claim 31, wherein: said preexisting additional functionality prevents circular evaluation (see for example, p.13, line 32, "It is never the case that the execution of a single *eval-when* expression will execute the body code more than once").

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 6, 8, 16, 30 and 36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Steele (Guy L. Steele, Common Lisp the language, 2nd edition) in view of Rodriguez (Rodriguez et al, Active Lisp Page)

Claim 6:

Steele discloses a method as in claim 1 above, but does not disclose said code for said program is XML code. However, Rodriguez in the same analogous art of using Lisp discloses an Active Lisp Pages which uses LISP in XML as a server scripting environment of creating dynamic pages and interactive applications over the Web (see for example, p.3, right col., lines 34-45, "XML Syntax"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Steele's LISP language in ALP and further embed with well-formed XML as disclosed by Rodriguez. One would have been motivated to do so to take advantage of well-formed XML and ALP server page engine solution as once suggested by Rodriguez. (see for example, p.1, left col., line 31 – right col., line 53, "Motivation" of ALP and also see p.3, right col., lines 34-45, "XML Syntax", "The advantage of having well-formed XML is the use that one can make of many available tools for such format, such as validators, schemas, and XSLT stylesheets.")

Claim 8:

Steele discloses a method as in claim 1 above, but does not disclose the method for implementation of client-server service. However, Rodriguez in the same analogous art of method for using LISP in ALP discloses the method comprising:

- Receiving a request for content via a network (see for example, p.1, Abstract, lines 14-15, "a client requests an ALP file/page from the server.");
- Transmitting said code with said additional functionality to a client via said network (see for example, p.1, Abstract, lines 23-24, "The server transmits standard HTML to the browser."); and

- Executing said code with said additional functionality at said client (see for example, p.1, Abstract, lines 25-26, "…only the result if the script is returned to the browser.")

Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Steele's LISP language in ALP to provide Web service. One would have been motivated to do so to provide secure service with simple, efficient advantages of ALP as once suggested by Rodriguez. (see for example, p.1, left col., line 31 – right col., line 53, "Motivation" of ALP)

Claim 16:

Steele discloses a method as in claim 13 above, but does not disclose said code for said program is XML source code. However, Rodriguez discloses an Active Lisp Pages which uses LISP in XML as a server side scripting environment of creating dynamic pages and interactive applications over the Web (see for example, p.3, right col., lines 34-45, "XML Syntax"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Steele's LISP language in ALP and further embed with well-formed XML as disclosed by Rodriguez. One would have been motivated to do so to take advantage of well-formed XML and ALP server page engine solution as once suggested by Rodriguez. (see for example, p.1, left col., line 31 – right col., line 53, "Motivation" of ALP and also see p.3, right col., lines 34-45, "XML Syntax", "The advantage of having well-formed XML is the use that one can make of many available tools for such format, such as validators, schemas, and XSLT stylesheets.").

Claim 30:

Steele discloses a method as claim 26 above, but does not disclose the method for implementation of client-server service. However, Rodriguez in the same analogous art of method for using LISP in ALP discloses the method comprising:

- Requesting said code by an Internet client (see for example, p.1, Abstract, lines 14-15, "a client requests an ALP file/page from the server.");
- Transmitting said code with said additional functionality to said Internet client after said step of automatically providing (see for example, p.1, Abstract, lines 23-24, "The server transmits standard HTML to the browser."); and
- Executing said code with said additional functionality using said Internet client (see for example, p.1, Abstract, lines 25-26, "...only the result if the script is returned to the browser.").

Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Steele's LISP language in ALP to provide Web service as disclosed by Rodriguez. One would have been motivated to do so to provide secure service with simple, efficient advantages of ALP as once suggested by Rodriguez. (see for example, p.1, left col., line 31 – right col., line 53, "Motivation" of ALP).

Claim 36:

Steele discloses a method in LISP to evaluate expression comprising the steps of:

- accessing code that includes an expression defining a first variable, said expression is dependent on a changeable item (see for example of implementation a *defun* form, p.14, line 14, a first variable "x" and expression); and
- automatically providing preexisting additional functionality to said code, said preexisting additional functionality evaluates said expression when said item changes and updates said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25 and related text explanation).

But does not explicitly disclose said apparatus that provides these service. However, Rodriguez in the same analogous art of method for using LISP in ALP discloses one or more processors in communication with said processor readable storage device, said one or more processors perform a method as Steele disclosed. (see for example, p.1, lines 11-13, "Web server", and "ALP processor"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Rodriguez's ALP server with processor and storage device embodied to perform Steele's method. One would have been motivated to do that, because it is efficient, simple and reusable. (see for example, p.1, left col., line 31 – right col., line 53, section Motivation, "Reusable components", "Efficiency", "Simplicity and fast development")

9. Claim 9-12, 22, 23, 25, 31, 32 and 34 are rejected under 35 U.S.C. 103(a) as being unpatentable over Steele (Guy L. Steele, Common Lisp the language, 2nd edition)

Claims 9-12:

Claims 9-12 are processor readable storage devices having processor readable code embodied on them, which are the product version of the claimed methods discussed as in claims 1-3 and 7 above. It is well known in the computer art to practice and store the computer readable code in such computer readable storage devices. Therefore, these claims are also obvious over Steele.

Claims 22, 23 and 25:

Claims 22, 23 and 25 are processor readable storage devices having processor readable code embodied on them, which are the product version of the claimed methods discussed as in claims 18, 19 and 21 above. It is well known in the computer art to practice and store the computer readable code in such computer readable storage devices. Therefore, these claims are also obvious over Steele.

Claims 31, 32 and 34:

Claims 31, 32 and 34 are processor readable storage devices having processor readable code embodied on them, which are the product version of the claimed methods discussed as in claims 26, 27 and 29 above. It is well known in the computer art to practice and store the computer readable code in such computer readable storage devices. Therefore, these claims are also obvious over Steele.

10. Claim 17 is rejected under 35 U.S.C. 103(a) as being unpatentable over Steele (Guy L. Steele, Common Lisp the language, 2nd edition) in view of Hickey (Hickey et al., LISP – a Language for Internet Script and Programming).

Claim 17:

Steele discloses a method as in claim 13 above, but does not disclose said code for said program is object code. However, Hickey discloses a method to implement LISP Applets which is object code and embed into web pages (see for example, p.7, section 3, LISP Applets, line 32 "Java byte code class file"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Steele's LISP language including special form eval-when to create LISP Applet for web pages. One would have been motivated to use LISP creating LISP applet object code which Hickey discloses to provide more efficient applets and greatly decreased download times and also be able to make use of the latest Java compilation technology. (see for example, p.7, lines 37-39, "potential to provide more efficient applets and greatly decreased download times. Another advantage of this approach is that by compiling to Java, we are able to make use of the latest Java compilation technology.").

11. Claims 20, 24, 28 and 33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Steele (Guy L. Steele, Common Lisp the language, 2nd edition) in view of Haible (Haible et al., Implementation Notes for GNU CLISP)

Claim 20:

Steele discloses a method as in claim 19 above, wherein: said additional functionality includes code that adds said first function to an object for said first variable, but does not explicitly disclose said code that provides a pointer to said first function to an object for said changeable item to be called by said object for said changeable item when said changeable item changes. However, Haible in the same analogous art of LISP implementation discloses that using "Weak Pointers" and "Foreign Pointers" to implement LISP in CLISP project (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "A weak pointer is an object holding a reference to a given object"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Haible's weak pointer to implement Steele's example. One would have been motivated to do so to quick access reference and without keeping the latter from being garbage-collected. (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "without keeping the latter from being garbage-collected.")

Claim 24:

Claim 24 discloses processor readable code embodied on processor readable storage devices, which can be executed by processor to perform the method as in claims 20 above for evaluating expressions. Therefore, this claim is also unpatentable by Steele and Haible, as it is well known in the computer art to store such methods in a processor readable device.

Claim 28:

Steele discloses a method according to claim 27, wherein: said additional functionality includes code that adds said first function to an object for said first variable, but does not explicitly disclose said code that provides a pointer to said first function to an object for said changeable item to be called by said object for said changeable item when said changeable item changes. However, Haible in the same analogous art of LISP implementation discloses that using "Weak Pointers" and "Foreign Pointers" to implement LISP in CLISP project (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "A weak pointer is an object holding a reference to a given object"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Haible's weak pointer to implement Steele's example. One would have been motivated to do so to quick access reference and without keeping the latter from being garbage-collected. (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "without keeping the latter from being garbage-collected.")

Claim 33:

Claim 33 discloses processor readable code embodied on processor readable storage devices, which can be executed by processor to perform the method as in claims 28 above for evaluating expressions. Therefore, this claim is also unpatentable by Steele and Haible, as it is well known in the computer art to store such methods in a processor readable device.

12. Claim 37 is rejected under 35 U.S.C. 103(a) as being unpatentable over Steele (Guy L. Steele, Common Lisp the language, 2nd edition) in view of Rodriguez (Rodriguez et al, Active Lisp Page) and further in view of Haible (Haible et al., Implementation Notes for GNU CLISP)

Claim 37:

Steele and Rodriguez disclose an apparatus according to claim 36 and Steele further discloses, wherein:

- Said expression is part of a constraint for said first variable (see for example code of implementation to expand a defun form into eval-when, p.14, line 17, "compiler::notice-function 'bar ' (x)");
- Said step of automatically providing includes creating an object for said constraint, adding a first function to said object that sets said first variable, determining dependency of said expression and adding a second function for said dependency to said object (see for example code of implementation to expand a defun form into eval-when, p.14, line 16-25, "lambda (x)", "lambda () (+ x 3)" and related text explanation); and

But both of them do not disclose said additional functionality includes code that adds said first function to an object for said first variable and code that provides a pointer to said first function to an object for said changeable item to be called by said object for said changeable item when said changeable item changes. However, Haible in the same analogous art of LISP implementation discloses that using "Weak Pointers" and "Foreign Pointers" to implement LISP in CLISP project (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "A weak pointer is an object holding a reference to a given object"). Therefore, it would have been obvious to one having ordinary skills in the art at the time the invention was made to use Haible's weak pointer to implement Steele's example and use Rodriguez's ALP server . One would have been motivated to do so to quick access reference and without keeping the latter from being garbage-collected. (see for example, p.5, section: extensions 1.6, Weak Pointers, line 22, "without keeping the latter from being garbage-collected.")

Conclusion

13. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

- Denis Attal (US 5860010) discloses the user of language with similar representation for programs and data.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Zheng Wei whose telephone number is (571) 270-1059. The examiner can normally be reached on Monday-Thursday 8:00-15:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature of relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571- 272-1000.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

ZW

TUAN DAM
SUPERVISORY PATENT EXAMINER